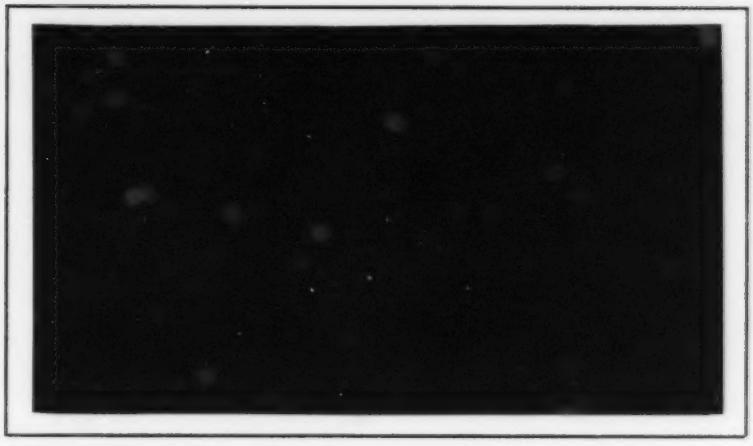

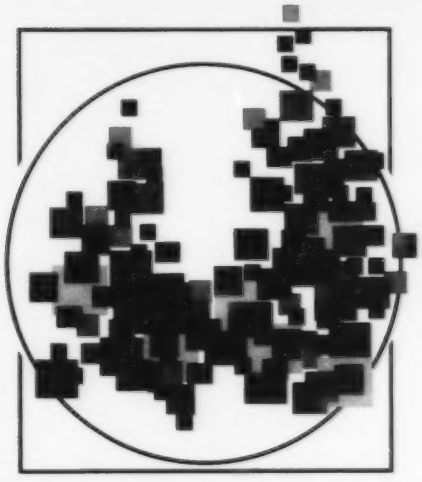

CENTRE FOR
SYSTEMS
SCIENCE

Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 / (604) 291-4704

92-05740/

Echidna Constraint Reasoning System: Next-generation Expert System Technology

William S Havens,Susan Sidebottom,Greg Sidebottom
John Jones,Miron Cuperman,Rod Davison

CSS-IS TR 90-09

Copyright © 1990
Expert Systems Lab

December 1990

Echidna Constraint Reasoning System: Next-generation Expert System Technology

William S. Havens, Susan Sidebottom, Greg Sidebottom,
John Jones, Miron Cuperman, Rod Davison

CSS-IS Technical Report 90-09

Expert Systems Laboratory
Centre for Systems Science
Simon Fraser University
Burnaby, British Columbia
Canada V5A 1S6

email: havens@cs.sfu.ca, sue@cs.sfu.ca

Abstract

Echidna is a new type of constraint logic programming system for model-based expert systems applications. The system improves upon the limitations of existing shells by combining aspects of schema-based knowledge representations, constraint logic programming (CLP) and dataflow dependency backtracking to produce a next-generation expert system shell. Echidna builds on recent advances in CLP by integrating within the language a clausal reasoning maintenance system and object-oriented knowledge structures. It is argued that next-generation expert systems will incorporate richer structured knowledge representations based on object-oriented programming principles and rely on more efficient constraint propagation and dependency backtracking control structures. Echidna demonstrates that these capabilities can be combined into a coherent reasoning system. A first version of the system is complete and being evaluated on a number of applications. This paper describes the architecture of Echidna as well as several applications under development.

This paper has been submitted for publication to the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, 1991 August 24.

1. Introduction

The expert system enterprise has been very successful. A multitude of expert system applications have been constructed. Although the underlying technology is mature, it still has inherent limitations: rule-based systems provide poor knowledge structuring for complex tasks; object-oriented systems address this inadequacy by providing powerful knowledge abstractions but lack a solid deductive foundation; hybrid systems combine rules and objects, but are often integrated in an *ad hoc* way resulting in an unclear semantics.

Logic programming languages, such as Prolog, have been recognized as a good basis for new expert system technology. Their precise semantics, relational form, and powerful pattern matching capabilities make logic programming languages ideal expert system development tools. Object-oriented knowledge representations are being incorporated into logic programming [CW88, Conn88]. Through advances in compiler technology, Prolog can now be implemented very efficiently [Warr77]. However, Prolog still suffers from the thrashing behavior which is characteristic of backtrack search [Mack77]. Two powerful techniques that ameliorate this problem are constraint processing and dependency-directed backtracking. Constraint processing techniques are integrated into logic programming in *constraint logic programming* (CLP), a new active research area [JL87]. Dependency backtracking has been shown to offer much more efficient search than traditional chronological backtracking [BP84; Drak88; Stal77]. Echidna is a new type of expert system shell which provides a coherent synthesis of object-oriented representations, constraint logic programming and data-driven dependency backtracking.

2. Current Expert System Technology

Sections 2.1 and 2.2, respectively, describe current expert system tools and logic programming technology from which Echidna is derived. The weaknesses of these technologies are noted.

2.1 Expert System Tools

There are more than 300 expert system shells available commercially today [Geva87]; many of these are rule-based. Rule-based methodology is mature, it has

had many successes, but its limitations are now apparent. There are two major problems: lack of descriptive adequacy (in the structure of the knowledge representation) and lack of procedural adequacy (in the methods for applying the knowledge) [HM83].

Expert system tools can be broadly classified as rule shells, object-oriented languages, and hybrid tools which combine rules and objects with other useful features.

2.1.1 Rule Shells

Simple rule shells, such as OPS5 [BFKM], represent knowledge in a relatively simple way by sets of rules, and match the antecedents of the rules against a relatively unstructured working memory. Using *modus ponens*, rules with matching antecedents are applied in a forward direction to add facts to the working memory (called *forward chaining*). More sophisticated rule shells also allow rules to be applied in the backward direction (called *backward chaining*). The simplicity of the representation and control mechanisms is appealing but deficient.

The knowledge base is a set of rules and facts, each representing a self-contained unit of knowledge. The result is a flat, unstructured knowledge base [HMM88]. This shallow knowledge representation leads to problems in both descriptive and procedural adequacy. The ontology of the task is implicit in the relationships among rules. The knowledge base does not reflect any structure inherent in the task itself. Consequently, the application of the rule knowledge cannot make use of this implicit structure to efficiently search for a solution. More sophisticated rule shells address these problems by providing focus mechanisms and priority rule execution.

Focus mechanisms organize rules into modules concerned with some particular aspect of a problem solution. The modules correspond roughly to objects in the task domain. The modules allow rules to be swapped in and out of working memory as consequences of other rule deductions. While more powerful than simple rule interpretation, focus mechanisms assume higher-order logics and much of the declarative reading of the rule semantics is lost. Likewise, rule priority schemes are also problematic. Rules are assigned priorities either statically (based on some *a priori* estimate of their utility) or dynamically during deduction (based on some external updating scheme). Again, there may result some improvement in

efficiency, but the descriptive adequacy of these techniques is not demonstrated. Both rule focus and priority mechanisms are a step back towards procedural programming, away from declarative adequacy, which rule based programming is intended to achieve.

2.1.2 Object-oriented Languages

Structured knowledge representation schemes include *frame systems* [Char75; Mins75; Wino75], *scripts* [Scha75], *plans* [Abel75] and *schemata* [Have83; RO76] as well as related *semantic network* representations [Hend75; Sowa84; Wood75]. In order for any of these formalisms to constitute a descriptively adequate language, their semantics must be formally specified. Hayes [Haye81] has criticized frame systems from this perspective. Woods [Wood75] analyzed the semantics of semantic networks and others have made these semantics more precise [Brac79; CS75; Marr82; Schu75].

These approaches provide powerful knowledge structuring mechanisms which allow objects to be organized into composition and specialization hierarchies which are isomorphic to the structure of the task domain. The key form of inference is inheritance down the specialization hierarchy. These systems usually contain a procedural programming language.

Object-oriented knowledge representations are widely used [BW77; CW88; Mins75]. Related model-based reasoning methods are now central to many expert systems [Davi84; Gene84]. An object-oriented knowledge representation which is on firm semantic ground with no procedural aspect is essential to the next-generation of expert system technology. Echidna provides such a system.

2.1.3 Hybrid Tools

Most large expert system development tools such as ART, KEE, Knowledge Craft and Nexpert Object are hybrid tools which, to some extent, remove the procedural aspect of object-oriented languages by combining objects (frame/schema representation) with rules [HMM88]. These systems also provide a mixed bag of other AI techniques including blackboards, truth maintenance systems, demons, and user interface tools. Unfortunately, these systems often graft techniques

together in an unprincipled way, resulting in unclear semantics for the system as a whole.

Combining rules and demons is a typical example of this unnatural grafting of techniques. Demons (also called active values) are processes which invoke procedures or rules used to update the state of a frame instance as its slot values change. LOOPS [SB83] is an early system which uses demons. KEE, ART and Knowledge Craft also provide demons [HMM88]. Demons are frequently used to implement constraint processing in frame-based systems. This is an inelegant, primitive, low-level feature left up to the programmer and has questionable semantics within the language. A more natural method for propagating constraints is identified in section 2.2.1.

As a result of this conglomeration of techniques in hybrid tools, there is more than one way to solve a problem and it is often difficult to decide which approach is appropriate. Moreover, the interaction between different systems such as truth maintenance and rules is not clear, since these systems were designed separately. Thus, hybrid tools do not provide a coherent and well founded development environment. The need for a expert system shell which provides a structured knowledge representation integrated with a strong semantic foundation is evident.

2.2 Logic Programming

Logic programming is a promising foundation for new rule-based expert system shells. It has several desirable characteristics. Its strong semantic foundation is an important advantage [Lloy84]. Recent advances have provided efficient implementations for logic programming languages such as Prolog [Warr77], as well as richer environments than were previously available [Colm90]. Also, it is easy to prototype various supplemental mechanisms using the metaprogramming capabilities of the Prolog language. As a result logic programming languages have recently received much attention as languages for the development of expert systems (eg [Merr89]).

However, most logic programming languages, such as Prolog, rely on standard backtracking (depth-first search with chronological backtracking). Although standard backtracking does prune significant portions of the search tree, improving on simple generate-and-test, it still takes exponential time in the number of variables

in the worst case [Knut75]. Therefore standard backtracking as a control mechanism is not adequate for most real-world applications.

We examine two responses to the inefficiencies of current logic programming technologies:

- the use of active constraint reasoning techniques to detect failure in the search sooner, and
- dependency backtracking to avoid unnecessary computation when failure is detected.

Both avoid search and therefore improve efficiency. Echidna is the only logic programming language we know of which integrates both of these techniques in a coherent architecture.

2.2.1 Active Constraint Processing in Logic Programming

Logic programming can profitably incorporate recent developments in constraint reasoning while maintaining the soundness of the underlying logical system.

Constraint reasoning techniques can be roughly classified along a continuum with two poles. At one end, in the Constraint Satisfaction Problem (CSP) tradition [Mack87], many techniques manipulate an explicit representation of variable domains and treat constraint relations as unstructured subsets of the Cartesian product of the domains of variables involved. At the other end of the spectrum, in approaches based on symbolic constraint solving [LMM88], variables are unstructured symbolic objects and constraints are highly structured objects which the techniques manipulate. The techniques at the CSP end of the spectrum admit to a wider variety of solution methods which can be tailored to specific CSPs, while the techniques at the constraint solving end allow reasoning about variables with far richer domains, for which explicit variable domain representations may be infeasible or impossible.

Several authors have investigated techniques at the CSP end of the spectrum to alleviate thrashing behavior during backtrack search [Freu78; Mack77; Mont74; Walt72]. During constraint processing, constraints are applied locally to restrict domains to only values that can participate in the global solution to the problem being solved. This active use of constraints significantly reduces the search space.

Constraint logic programming (CLP) is an extension of logic programming to include efficient algorithms for specialized computational domains in a way that preserves the attractive properties that implementations of logic programming, like Prolog, possess. A CLP scheme defined in [JL87] formally specifies some semantic properties that all instances of the scheme possess. Many CLP systems, including Prolog, Prolog II [Colm84], CLP(R) [JM87] and CLP(Σ^*) [Wali89] are instances of this scheme. However, there has been a recent surge of CLP system development and the relationship between many systems and the CLP scheme has not been formally studied. Some systems, such as CHIP [VanH89] and Prolog III [Colm90], are based on different theoretical models. See Cohen [Cohe90] for a current review.

Instances of the CLP scheme and Prolog III are based on constraint solving techniques while CHIP provides both CSP and constraint solving techniques, depending on the domain of computation[1]. Echidna, like CHIP, is a CLP language which actively processes constraints using appropriate algorithms for different computational domains. Echidna processes constraints for three computational domains: arbitrary constraints on finite discrete sets, numeric constraints on sets of integers and numeric constraints on sets of reals; the respective algorithms used for the different domains is discussed in section three.

2.2.2 Dependency-directed Backtracking

Dependency-directed backtracking is the second response to the inefficiencies of chronological backtracking [KSS79; Stal77]. It provides two significant improvements for implementing nondeterministic search:

- backtracking to an appropriate choice, and
- avoiding rediscovery of contradictions.

When failure occurs during the search process, chronological backtracking simply undoes the most recent goal choice made. In contrast, dependency-directed backtracking looks for the most recent goal which may have been a cause of the

---

[1]'CHIP uses CSP techniques for finite domains, and symbolic constraint solving techniques for boolean terms (boolean unification) and rational terms (a simplex-like algorithm) [VH89 p.10].

failure (called the culprit). This is not necessarily the most recently solved goal. Once the culprit goal is established, all goals susequent to it are undone and the culprit retried for a different solution.

Dependency-directed backtracking avoids rediscovering contradictions and therefore avoids thrashing behavior. When a contradiction is discovered by the inference engine, the set of choices (called the nogood) which contributed to contradiction is saved in a nogood database. Every choice set in this database represents an environment which is known to fail and consequently need not be retried. When backtracking introduces a new choice to current state, the nogood database is checked before continuing.

Dependency backtracking has been effectively applied to logic programming, augmenting the control regime of Prolog by clausal reason maintenance system (RMS) techniques [BP84; Drak88]. Choices correspond to clauses selected by the interpreter to solve goals. Identifying the culprit for dependency backtracking requires keeping track of the variable dependencies among subgoals.

We extend dependency-directed backtracking in three significant ways. First, Echidna adapts clausal RMS techniques to a CLP language. Second, Echidna enables the use of persistent object variables. The increased complexity of reason maintenance for persistent constraints and persistent variables leads to a logical dataflow architecture for the reasoning engine.

Third, Echidna optimizes the reuse of clause instances on failure without incurring the expense of an ever increasing nogood database. In contrast, [Drak88] showed how to discard dependency information on backtracking thereby limiting nogood database size. We extend this notion and reuse failing clauses whenever possible.

3. Echidna as an Expert System Shell

Echidna provides a coherent synthesis of three technologies;

- a schema knowledge representation,
- a Prolog-compatible logic programming language which supports active constraints among objects, and
- a justification-type reason maintenance system for efficient dependency backtracking.

Echidna may be viewed as a new form of expert system shell which has evolved from previous work in the Platypus[2] constraint reasoning system [HR89; Have89]. The design of Echidna has been influenced significantly by constraint-based expert systems (e.g. [MDM86]) and by current work in CLP [Colm90; JL87], in particular by Van Hentenryck's CHIP system [VanH89]. Like CHIP, Echidna couples constraint propagation with an underlying logic programming language. Both languages inherit the first-order Horn clause semantics of Prolog and employ a *k*-ary arc consistency algorithm [Mack77] for constraint propagation over discrete domains. Echidna adds a breadth-first incremental derivation scheme and justification-based reasoning maintenance system (RMS) based on [Doyl79] for implementing true dependency backtracking. The RMS is a significant extension of the logic programming language RMSs used by Drakos [Drak88] and by Bruynooghe and Pereira [BP84; YW89]. In comparison, CHIP inherits both the depth-first search rule and chronological backtracking on failure endemic to Prolog.

Echidna also defines a straightforward formulation of object-oriented programming within logic programming. The realization of objects within logic programming has seen many proposals (for example [CW88, Conn88]). In Echidna, objects are represented as predicate schemata with persistent state. Objects are accessed by unifying goals (logical messages) with the predicates (logical methods) defined within the schema. Schema instances are considered legitimate logical terms which may be created, sent messages, passed as arguments and composed into networks, all via unification.

---

[2]Unlike Platypus, Echidna is a relational language. Platypus was implemented as a functional extension to Scheme.

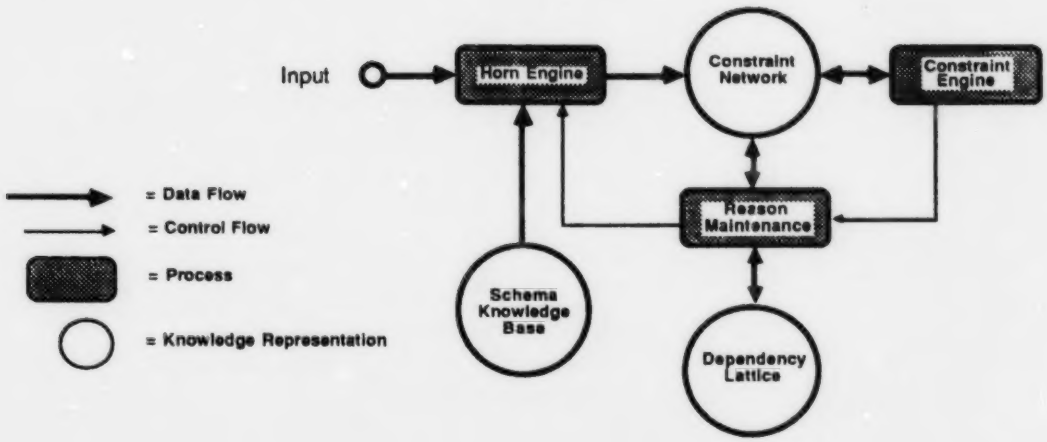

Figure 1 Echidna Overview

Figure 1 illustrates the overall system architecture. There are three major reasoning subsystems: the Horn engine, the constraint engine and the reason maintenance system. The Horn engine is a SLD-resolution [Lloy84] theorem prover which operates on the Horn clauses contained within the schemata. The responsibility of this engine is to construct instances of the schemata which satisfy the queries presented as input to the system.

The result is the constraint network where the nodes of the network are schema instances and the *k*-ary arcs are constraints asserted to hold between the parameters of related instances.[3] The constraint engine uses this representation directly to propagate *k*-ary arc consistency among instances thereby refining the network description produced by the Horn engine.

Echidna processes constraints in three computational domains: arbitrary constraints on finite discrete sets, numeric constraints on sets of integers and numeric constraints on sets of reals. Finite discrete constraints are presently processed using a modification of the AC3 algorithm [Mack77] for *k*-ary constraints. We are investigating processing integer constraints using a more efficient specialization of AC3 similar to the approach used in CHIP [VanH89] and Con-Pro [Side90].

---

[3]The CHIP system also produces a constraint network embedded within the derivation tree as a consequence of clause elaboration.

The processing of numeric real constraints is being introduced into the current implementation by representing variable domains explicitly as hierarchically structured sets of intervals and by propagating equalities and inequalities using a specialized hierarchical arc consistency algorithm based on [MMH85]. This approach differs from constraint solving approaches typically employed in CLP languages such as CHIP, Prolog III and CLP(R). First, it can process non-linear constraints and constraints involving non-arithmetic functions such as root extraction, exponentiation and trigonometric functions. Second, the accuracy to which variables represent solutions to the constraints can be incrementally increased using declarative accuracy control primitives. Other logic programming systems which provide real interval arithmetic constraint processing include *Logical Arithmetic* [Clear87] and *BNR Prolog* [BNR88]. Cleary's Logical Arithmetic has control primitives which force variables to be restricted to the resolution of the underlying machine number implementation and force choice points in the search with every increase in accuracy not detectable by the constraint processing algorithms. BNR Prolog allows programming of search depth by mixing logic and control. Echidna will be able to unleash accuracy incrementally according to declarative control rules separate from the logic of the program; choices will be represented explicitly and efficiently in hierarchical variable domains which contain sets of intervals.

In the background, the RMS records the choices made by the Horn engine and their consequences propagated by the constraint engine as events in the dependency lattice. The events are nodes in a straightforward justification-type reasoning maintenance system (RMS) [Doyl79]. Echidna uses the RMS to record causal changes to the derivation tree, not logical implication. Assumptions correspond to both user queries and the nondeterministic choices of clauses made by the Horn engine to solve goals. Events supported by these assumptions include variable instantiation by unification, schema instance creation, and constraint propagation. The partial order relation is causal support. The maximal nodes of the lattice are all the assumptions and the minimal nodes are the fringe of unification and constraint propagation.

The RMS associates a label [deKl86] with every node in the lattice. The label contains those assumptions which were used to obtain the derivation state represented by the node. Each of the assumptions in the label is either a user query

or a clause choice made by the Horn engine (or a free choice made by a generator such as split(X)[4]). On the occurrence of an inconsistent constraint or the failure of a predicate, the RMS considers the label associated with the failing node to be a nogood. One of the assumptions in the nogood will be identified as the culprit which is deemed to have caused the failure. If the culprit is a user query, deleting it will cause the query goal to be undone along with all its side effects. Otherwise, the culprit is a clause choice. Changing this choice will cause the Horn engine to attempt a different derivation. In either case, the action will necessarily remove the failure.[5]

The culprit selected for dependency backtracking is always selected such that the derivation of the Horn engine can be extended to a SLD-refutation [Lloy84] and its search is complete. To guarantee these qualities, we define a chronological ordering of all assumptions (e.g. preorder in the resolvent tree) with which to order the backtrack search. The culprit is always the most recent assumption in the nogood. All of the alternate choices for a given culprit are explored before an assumption of higher order is considered.

4. Applications

Several research projects which use Echidna as the core reasoning engine are currently under development at Simon Fraser University in the Expert Systems Laboratory. We review three applications here, considering the last in more detail:

1) an intelligent design system called *intelCAD*,
2) an intelligent graphics interface project called *IGI*, and
3) an automotive diagnosis system called *SuperEAS*.

Each is indicative of the need for model-based constraint reasoning. These complex applications would be extremely difficult or impossible to develop using a simple rule-based tool. We illustrate each application by describing its task, its natural ontology as a constraint-based hypothetical reasoning problem, the appropriateness

---

[4]This generator was introduced in [VanH89].

[5]Of course, the reason for the failure may manifest itself in another failure elsewhere in the lattice.

of schema knowledge representation, and the need for efficient constraint reasoning methods.

4.1 Intelligent Design

IntelCAD is an intelligent computer-aided design system which uses Echidna as its reasoning engine. Intelligent CAD is a new research field which will is expected to have a tremendous effect on the process of design [TY84]. Current CAD systems still operate on primitive representations of traditional mechanical drawings. Although recent parametric design systems have introduced richer object-oriented representations which support object models, parameters and rule-based selection of components, they still have inherent limitations. Ideally, mechanical engineers and architectural designers should work with a hierarchy of conceptual components from which they construct complex functional design artifacts. A rich knowledge representation is essential. The correct assembly of components must obey certain rules and constraints. This process may involve considerable hypothetical "what-if" reasoning to achieve a "correct" design. Echidna is able to provide this additional constraint-based hypothetical reasoning capability.

In intelCAD, design components are represented as schema classes which are parameterized by their design attributes. Complex components are specified as assemblies of parts connected by constraints among their respective parameters. Designs are created generatively by sending goals to schemata which are interpreted by the logical methods defined within. The result is a network of schema instances and their mutual constraints which represent the design. The design process continues as new design goals are derived thereby generating new instances and new constraints in the constraint network and consequently propagating constraints among the existing instances. When a goal fails, dependency backtracking is initiated to look for an alternative generative network composition which satisfies the specified goals.

4.2 Intelligent Graphics Interface

An intelligent graphics interface (IGI) system is being developed for supervisory control applications in Echidna. The function of the IGI is to circumvent the information bottleneck which exists between operators and real-time supervisory control systems. Current workstations and the attention limitations of human

operators impose a very small bandwidth on the display of information to the operators. This limitation can be overcome by using an IGI employing a model-based mediation paradigm. The IGI is the mediator between the operator and control system. It uses various specialized knowledge bases, such as a graphical user interface knowledge base and an application knowledge base, to both intelligently manage the display of information to the operator and assist the operator in supervising the application control system.

Since an operator naturally views a control system database as a (perhaps) complex set of physical or logical objects in the control system, a model-based representation is appropriate. For example, in an application of IGI being developed with the British Columbia Telephone Company (BCTel) long distance network, switch circuits are physical components of the system while the trunk routes which they support are logical entities. In IGI, both are represented as schemata in an Echidna knowledge base. Each object is characterized by a small number of parameters which specify the possible instantiations of the schema. Knowing these parameters for a particular instance of the schema fully represents the state of the object to the operator . The behavior of the object can be captured through Echidna methods and constraints. Methods specify the operations allowed on particular objects. Relationships are manifest as constraints between the parameters of related schemata.

Constraints are a natural part of the ontology for IGI applications. Constraints express relationships among task objects in both the application knowledge base and various graphical user interface knowledge bases. For example, in the BCTel application knowledge base, constraints express digital switch capacities, traffic flows and possible routing paths. Within the interface knowledge bases, constraints specify screen space allocations, alarm icon priorities and the use of color assignments within various display models.

Constraints manifest the dialogues established between models. When a dialogue link is connected between models, the ensuing dialogue establishes a new constraint. Whenever the state of either instance changes, it is communicated to the other thereby realizing bidirectional constraint propagation. The dialogue link may be persistent over time, providing a constraint between the schemata indefinitely. The model-based dialogue paradigm is clearly more powerful than simpler

message-passing methodology (although it could be simulated using repeated messages).

4.3 Automotive Diagnosis

Automotive engine diagnosis is an important application domain for expert systems. We are developing a model-based expert system for diagnosing engine computer control system malfunctions. The system uses a knowledge base of models of the major electrical and mechanical systems in the engine. The models provide a mixture of deep and shallow knowledge about the operation of the engine. The models must be competent to express the operating states that each engine component can assume when functioning both normally and abnormally. Each model has a set of default assumptions about its normal operating behavior. By manipulating these assumptions (as causal hypotheses), the model can be moulded to the actual state of the engine. The reasoning system must abductively search for the correct minimal set of assumptions which explains the model / engine differences.

We represent the engine as a causally connected network of components. Each component is defined as a schema having a small set of characteristic parameters, such as pressure or flow. The connections between components of the model are constraints over these parameters. For example, a rubber pneumatic hose is a component which provides constraints between the two other engine components it couples (e.g. the intake manifold and emission control system). It links the pressure and flow rates of these coupled components.

Associated with each schema are those assumptions which describe its possible functional states. There is a separate assumption for each possible qualitative component state which describes its behavior in that state. For example, a pneumatic hose can be normal (by default), split or blocked. It therefore needs to have at least three qualitative states:

State	Constraint
1) assumed normal operation:	input flow = output flow
2) assumed blocked:	input flow = output flow = 0
3) assumed leaking:	input flow > output flow

The model initially makes the first assumption by default but may assume alternative states in order to explain the observed symptoms. During diagnosis, a failure of the model to predict the actual observed behavior of the engine causes dependency backtracking to change its current assumption about some component (such as the hose) thereby changing its operational state. When every engine component has its internal state consistent with the observed behavior of the engine, then the corporate state of all components represents the computed diagnosis.

The Echidna schema definition in figure 2 shows a model for a generic hose. In our model, hoses are simple objects which connect with hose fittings and conduct fluids under pressure in either direction. They have three possible states: good, blocked or leaking. A good hose exhibits no fluid flow losses and has no significant frictional losses. A blocked hose has no flow and the fluid pressures at either end are not related. A leaking hose exhibits equal pressures but there is a significant fluid loss in the hose.

```
schema hose
 { [0.0..inf] InFlow.
   [0.0..inf] OutFlow.
   [0.0..inf] InPress.
   [0.0..inf] OutPress.
   fitting F1.
   fitting F2.

   new (F1, F2):-
      F1:flow(Inflow),
      F1:pressure(InPress),
      F2:flow(OutFlow)
      F2:pressure(OutPress),
      default.

   default :- good; blocked; leaking.

   good :- InPress =:= OutPress, InFlow =:= OutFlow.
   blocked :- InFlow =:= 0, OutFlow =:= 0.
   leaking :- OutFlow < InFlow, OutPress  =:= InPress.

   ...
   }
```

Figure 2 Simple schema for a Hose in an Automobile Engine

The Echidna code in figure 2 defines a hose schema. The schema has six parameters which are persistent instance variables. The first four, Inflow, Outflow, InPress, and OutPress, have domains in the positive real numbers; the last two, F1 and F2, are instances of the fitting schema. A new hose instance is created by sending the message "new" to the hose schema. The associated method in the schema receives the two hose fitting instances, F1 and F2, to connect the hose instance. The method then establishes both fluid flow and pressure constraints between the hose and its fittings by sending appropriate constraints to these instances. Finally, the method calls the default method which disjunctively selects "good" as its first choice for the internal state of the instance. In turn, the "good" method establishes that the flow into the hose equals the flow out and that the pressure at either end is the same. On failure, the hose will next choose state

"blocked" which establishes that there is no fluid flow in or out of the hose. On a subsequent failure, the hose assumes the state "leaking", in which the pressure is the same at both ends of the hose, but the flow out of the hose is less than the flow into the hose. This set of three model states is competent since it describes all possible modes of operation for the hose, both normal and abnormal.

5. Conclusion

Expert system technology is mature but current shells still have many inherent limitations. Echidna is a next-generation expert system shell which addresses this problem by bringing together current research in logic programming, constraint reasoning, model-based recognition and object-oriented programming.

In this paper we have described the Echidna system and how it is suited to solve three complex applications. These applications show the power of Echidna's constraint-based hypothetical reasoning and schema knowledge representation facilities. Since these techniques are integrated into a coherent reasoning architecture, we believe Echidna to be a true next-generation expert system tool. The first implementation is now operational and being evaluated. A second implementation is under development.

6. Acknowledgements

The IGI project is supported by a research grant from PRECARN Associates Ltd. Toronto, Canada and is jointly being pursued by Simon Fraser University and Microtel Pacific Research Lt d. IntelCAD is supported by research grants from the BC Science Council and an NSERC CRD grant and has two industrial sponsors: Pan Abode of Richmond and Microlight corporation of Vancouver. The automobile diagnosis system is being developed with support from McCarney Technologies Ltd.of Vancouver. Research support is also being provided by the Natural Sciences and Engineering Research Council of Canada.

References

[Abel75] Abelson, R. P. (1975) "Concepts for Representing Mundane Reality in Plans." In *Representation and Understanding*, D. G. Bobrow and A. Collins (ed.), Academic Press, New York, NY, pp. 273-309.

[BNR88] Bell-Northern Research Ltd. (1988) BNR Prolog User Guide.

[BW77] Bobrow, D. G. and T. Winograd (1977) "An Overview of KRL: A Knowledge Representation Language," *Cognitive Science*, **1**, pp. 3-46.

[Brac79] Brachman, R. J. (1979) "On the Epistemological Status of Semantic Networks." In *Associative Networks*, Academic Press, New York, NY, pp. 3.

[BFKM85] Brownston, L., R. Farrel, E. Kant and N. Martin (1985), "Programming Expert Systems in OPS5" Addison-Wesley, Reading, MA.

[BP84] Bruynooghe, M. and L. M. Pereira (1984) "Deduction Revision by Intelligent Backtracking." In *Implementations of Prolog*, J. A. Campbell (ed.), Ellis Horwood Publishers, Chichester, England.

[CS75] Cercone, N. and L. Schubert (1975) "Towards a State-based Conceptual Representation." In Proc. *International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, pp. 83-90.

[Char75] Charniak, E. (1975) "Organization and Inference in a Frame-like System of Knowledge." In Proc. *Theoretical issues in Natural Language Processing*, Cambridge, MA.

[CW88] Chen, W. and D. S. Warren (1988) "Objects as Intensions." In Proc. *5th Int. Comp & Symp. on Logic Programming*, MIT Press, Cambridge, Mass.

[Clea87] Cleary, J. G. (1987) "Logical Arithmetic," *Future Computing Systems*, **2** (2), pp. 125-149.

[Cohe90] Cohen, J. (1990) "Constraint Logic Programming," *Communications of the ACM*, **33** (7), pp. 52-68.

[Colm90] Colmerauer, A. (1990) "An Introduction to Prolog III," *Communications of the ACM*, **33** (7), pp. 69-90.

[Colm84] Colmerauer, A. (1984) "Equations and Inequations on Finite and Infinite Trees." In Proc. *Fifth Generation Computer Systems*, pp. 85-99.

[Conn88] Connery, J.S. (1988) "Logical Objects" In Proc. *5th Int. Comp & Symp. on Logic Programming*, MIT Press, Cambridge, Mass.

[Davi84] Davis, R. (1984) "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, **24** (3).

[deKl86] de Kleer, J. (1986) "An Assumption-based TMS," *Artificial Intelligence*, pp. 127-162.

[KSS79] de Kleer, J., J. Doyle and G. L. Steele (1979) "Explicit Control of Reasoning." In *Artificial Intelligence: An MIT perspective*, P. H.

Winston and R. H. Brown (ed.), MIT Press, Cambridge, MA, pp. 94-116.

[Doyl79] Doyle, J. (1979) "A Truth Maintenance System," *Artificial Intelligence*, **12**, pp. 231-272.

[Drak88] Drakos, N. (1988) "Reason Maintenance in Horn-Clause Logic Programs." In *Reason Maintenance Systems and Their Applications*, B. Smith and G. Kelleher (ed.), John Wiley & Sons, Toronto, pp. 77-97.

[Freu78] Freuder, E. C. (1978) "Synthesizing Constraint Expressions," *Communications of the ACM*, **21** (11), pp. 958-966.

[Gene84] Genesereth, M. R. (1984) "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence*, **24** (3).

[Geva87] Gevarter, W. B. (1987) "The Nature and Evaluation of Commercial Expert Systems Building Tools," *IEEE Computer*, **20** (5), pp. 24-41.

[HMM88] Harmon, P., R. Maus and R. Morrissey (1988) *Expert Systems Tools & Applications*, John Wiley & Sons, Inc., Toronto.

[Have83] Havens, W. (1983) "Recognition Mechanisms for Schema-based Knowledge Representations," *International Journal of Computers and Mathematics*, **9**, pp. 185-199.

[HM83] Havens, W. and A. Mackworth (1983) "Representing Knowledge of the Visual World," IEEE Computer, **16**(10) pp. 90-96.

[HR89] Havens, W. C. and P. S. Rehfuss (1989) "Platypus: a Constraint-Based Reasoning System." In Proc. *International Joint Conference on Artificial Intelligence(IJCAI-89)*, Detroit, Mich.

[Have89] Havens, W. S. (1989) "The Platypus Expert Systems Shell." In Proc. *Conf. of Knowledge-Based Computer Systems*, Bombay, India.

[Haye81] Hayes, P. (1981) "The Logic of Frames." In *Readings in Artificial Intelligence*, B. Webber and N. Nilsson (ed.), Tioga Publishing, Palo Alto, CA, pp. 451-458.

[Hend75] Hendrix, G. (1975) "Expanding the Utility of Semantic Networks through Partitioning." In Proc. *International Conference on Artificial Intelligence (IJCAI-75)*, Tbilisi, USSR, pp. 115-121.

[JL87] Jaffar, J. and J.-L. Lassez (1987) "Constraint Logic Programming." In Proc. *Fourteenth ACM POPL Conf.*, Munich.

[JM87] Jaffar, J. and S. Michaylov (1987) "Methodology and Implementation of a CLP System." In Proc. *Fourth International Conference on Logic Programming*, Melbourne, Australia.

[Knut75] Knuth, D. E. (1975) "Estimating the Efficiency of Backtrack Programs," *Mathematics of Computation*, **29** (129), pp. 121-136.

[LMM88] Lassez, J.-L., M. J. Maher and K. Marriot (1988) "Unification Revisited." In *Foundations of Deductive Databases and Logic Programming*, J. Minker (ed.), Morgan Kaufmann, Los Altos, CA, pp. 746.

[Lloy84] Lloyd, J. W. (1984) *Foundations of Logic Programming*, Springer-Verlag, New York.

[Mack77] Mackworth, A. K. (1977) "Consistency in Networks of Relations," *Artificial Intelligence*, **8**, pp. 99-118.

[Mack87] Mackworth, A. K. (1987) "Constraint Satisfaction." In *Encyclopedia of Artificial Intelligence*, S. C. Shapiro (ed.), John Wiley & Sons, pp. 205-211.

[MF85] Mackworth, A. K. and E. C. Freuder (1985) "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence*, **25**, pp. 65-74.

[MMH85] Mackworth, A. K., J. A. Mulder and W. S. Havens (1985) "Hierarchical arc consistency: exploiting structured domains in constraint satisfaction problems," *Computational Intelligence*, **1**, pp. 118-126.

[Marr82] Marr, D. (1982) *Vision*, W.H. Freeman, San Francisco, CA.

[Merr89] Merritt, D. (1989) *Building Expert Systems in Prolog*, Springer-Verlag, New York.

[Mins75] Minsky, M. (1975) "A Framework for Representing Knowledge." In *The Psychology of Computer Vision*, P. Winston (ed.), McGraw-Hill Book Company, New York, NY.

[MDM86] Mittal, S., C. L. Dym and M. Morjaria (1986) "PRIDE: An Expert System for the Design of Paper Handling Systems," *Computer*, **July**, pp. 102-114.

[Mont74] Montanari, U. (1974) "Networks of Constraints: Fundamental Properties and Application to Picture Processing," *Information Sciences*, **7**, pp. 95-132.

[RO76] Rumelhart, D. E. and A. Ortony (1976) "The Representation of Knowledge in Memory." TR-55, *Center for Human Information Processing, Department of Psychology, University of California at San Diego, LaJolla, CA*, 1976.

[Scha75] Schank, R. C. (1975) "The Structure of Episodes in Memory." In *Representation and Understanding*, D. G. Bobrow and A. Collins (ed.), Academic Press, New York, NY, pp. 237-272.

[Schu75] Schubert, L. (1975) "Extending the Expressive Power of Semantic Networks." In Proc. *International Joint Conference on Artificial Intelligence (IJCAI-75)*, Tbilisi, USSR, pp. 158.

[Side90] Sidebottom, G. (1990) "Satisfaction of Constraints on Non-negative Integer Arithmetic Expressions", Open File Report # 1990-15, Alberta Research Council, July 1990.

[Sowa84] Sowa, J. F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publishing Co., Reading, MA.

[Stal77] Stallman, R. and G. Sussman (1977) "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, **9**, pp. 135-196.

[SB83] Stefix, M. and D. Bobrow (1983) "The LOOPS manual.", *Xerox Palo Alto Research Center*, 1983.

[TY84] Tomiyama, T. and H. Yoshikawa (1984) "Requirements and Principles for Intelligent CAD Systems." In Proc. *IFIP WG 5.2 Working Conference*, North Holland, pp. 1-28.

[VanH89] Van Hentenryck, P. (1989) *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge.

[Wali89] Walinsky, C. (1989) "Constraint Logic Programming with Regular Sets." In Proc. *Sixth International Conference on Logic Programming*, Lisbon, Portugal, pp. 181-196.

[Walt72] Waltz, D. L. (1972) "Generating Semantic Descriptions from Drawings of Scenes with Shadows." Technical Report, *MIT*.

[Warr77] Warren, D. H. D. (1977) "Implementing Prolog - Compiling Predicate Logic Programs." D.A.I. Research Report 39, *University of Edinburgh*, May 1977.

[Wino75] Winograd, T. (1975) "Frame Representations and the Procedural-declarative Contraversy." In *Representation and Understanding*, D. G. Bobrow and A. Collins (ed.), Academic Press, New York, NY, pp. 185-210.

[Wood75] Woods, W. A. (1975) "What's in a Link: Foundations for Semantic Networks." In *Representation and Understanding*, D. G. Bobrow and A. Collins (ed.), Academic Press, New York, NY, pp. 185-210.

[YW89] You, J.-H. and B. Wong (1989) "Intelligent Backtracking in Prolog Made Practical" Technical Report TR-89-4, Department of Computing Science, U. Alberta, Edmonton, Alberta, Canada.

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

•CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.